Design and Analysis of *Hincent*, Quick Content Distribution With Priorities and High Incentives

Debessay Fesehaye Kassa
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801-2302
Email: dkassa2@illinois.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801-2302
Email: klara@illinois.edu

***Abstract*—Existing literature shows that Peer-to-Peer (P2P) content sharing can result in significant scalability gains in addition to assisting content distribution networks (CDNs). However, currently proposed CDN and P2P hybrid schemes do not provide accurate and efficient incentives to attract and maintain more peers. Besides, they do not use efficient prioritized congestion control and content source selection mechanisms to reduce content transfer time.**

We present *Hincent*, a quick content distribution protocol, which uses efficient prioritized rate allocation and content selection algorithms offering high incentives to participating peers. The fair incentives attract more peers which securely download and distribute contents. This in turn can benefit content providers and network operators. The *Hincent* rate allocations results in quicker content transfer time when compared with existing schemes. *Hincent* also employs effective rate enforcement mechanisms without requiring changes to the TCP/IP stack or to existing routers. Unlike existing centralized schemes such as YouTube, the design allows peers to have full control of (their) contents while sharing them with others using personal web servers. We have also briefly describe how *Hincent* can be implemented using surrogate (cloud or cloudlet) servers with OpenFlow vSwitches.

We have implemented *Hincent* in the NS2 simulator. Our detailed trace-based experiments show that *Hincent* outperforms existing schemes in terms of file download time and throughput by up to 30% on average. The results also demonstrate that *Hincent* obtains fair uplink prices for the uploaders and fair cost for the downloaders maintaining an overall system fairness. Besides, the results show the efficient *Hincent* enforcements of the prioritized allocations. Our *Hincent* implementation using an Apache SQL Server with PHP in Linux virtual machines demonstrates that *Hincent* content index management mechanisms are scalable.

I. Introduction

With the fast growth of the Internet and networking technologies, there has been an explosive growth of online content [1], [2]. These online contents are generated either by centralized content providers (Comcast, Amazon, etc) or distributed users (Youtube, Facebook, etc). Such content generation is expected to grow even more (40-45% a year) [2] with the further expansion and sophistication of the Internet and networking technologies.

Traditionally, centralized content providers (CCP) use content distribution networks (CDN) to distribute their contents to their customers. With the increase in high bandwidth content demands [1], content providers should either over-provision their bandwidth to handle peak demands or rely on purchased service such as Akamai. However as discussed in [3] it is cheaper for content providers to purchase bandwidth from their users than using third party content distribution networks (CDNs) or purchasing the infrastructure to directly serve contents. Besides assisting CDNs, using P2P networks results in significant scalability gains as discussed in [4], [5].

While using cooperative customer peers to distribute content, providers need to be mindful about incentives to pay back peers for their upload bandwidth. Besides, content providers need to make sure that the incentives and returns are accurate enough to offer better quality of service (QoS) guarantees. Using an efficient, fair and accurate peer incentive mechanism can also benefit content providers and network operators significantly. Content providers can save on bandwidth cost by buying peer link bandwidth. Besides, peers who get significant credit (financial or content credit) from uploading content are most likely to subscribe to more contents potentially increasing the content demand. More content demand can also translate into more link bandwidth demand which can benefit network operators. As discussed in [6] distributed user generated contents can also be feasibly shared from homes allowing users (peers) full ownership and control of their contents.

Existing incentive-based content sharing mechanisms such as Price-Assisted Content Exchange (PACE) [7], [8] and Dandelion [3], [9] do not use efficient incentive mechanisms. For instance PACE does not guarantee fair-exchange of content for payment. Dandelion uses fixed bandwidth pricing mechanism that peers do not decrease their prices to attract more customers when they have high upload rate and vice-versa. Besides, such existing schemes do not find and enforce accurate rates at which peers can download content from other peers so as to minimize content transfer time. They do not give a mechanism to prioritize content transfers which is an important component of 3D [10] and other multi-view streaming applications where some streams are more important than others based on the view angle. Besides, existing work does not provide an efficient content source selection mechanism which chooses a source that leads to high throughput and low file completion time.

In this paper we present *Hincent*, an efficient *prioritized distributed cross-layer content routing and congestion control* protocol with *high incentives* to the participating peers. The design of *Hincent* enables distributed network peers to securely exchange content by providing high monetary and bandwidth incentives for their resource (bandwidth, storage, energy, processing, etc) used in the content transfer. It allows users to have full control of their contents which can be a 2D, 3D data or ordinary file. *Hincent* can limit the lifetime of the content to a user-defined parameter. This content age and the prioritized rate allocation features of *Hincent* are specially important for 2D and 3D live streaming contents which have real time requirements. For instance, to render a 3D video, streams should be synchronized and rendered within a short time gap between them. The fair and accurate incentive, rate allocation, enforcement and content source selection mechanisms of *Hincent* allows peers to exchange content with smaller transfer time than existing schemes. The *Hincent* protocol does not need changes to the TCP/IP stack and existing network devices (routers, switches) that it can be easily deployed in the current Internet.

We also discuss an extension of *Hincent* using surrogate cloud or cloudlet servers to help peer clients transfer contents faster than using existing schemes. The servers are equipped with OpenFlow vSwitches and form a network. These servers in the network are connected using either dedicated or overlay links. Cloudlets [11] are decentralized and widely-dispersed Internet infrastructure whose compute cycles and storage resources can be leveraged by nearby mobile computers.

We have implemented *Hincent* in the NS2 [12] simulator and using an Apache SQL server with PHP in Linux virtual machines. The NS2 simulator is so robust that descriptions of the streams of the 3D content can be taken as inputs to produce an emulated 3D video as output. The simulation results show how *Hincent* can outperform existing content distribution schemes in terms of download time and throughput. The results also demonstrate that the different components of *Hincent* work according to the design. The SQL implementation of *Hincent* using PHP shows that *Hincent* can scale to millions of peers and contents.

In [13] we presented a short version of our *Hincent* work. In that short version, among other things, we did not show how *Hincent* rate allocation can be TCP friendly, there was no discussion of multiple server selection policies and there was no extension of *Hincent* using surrogate servers. Besides, the short version did not discuss the *Hincent* content index management (CIM) schemes and how the schemes scale. We also did not present Apache SQL server implementation experiments of *Hincent* CIM in our short version.

The main contributions of this work are as follows.

- We have designed an efficient content distribution protocol (*Hincent*) with cross-layer content routing (content source selection) and congestion control mechanisms. It can allow distributed users (peers) to have full control of their contents while securely sharing them.
- We have shown that *Hincent* provides accurate and efficient incentive mechanisms to benefit content providers, content users and network operators. The incentive is in real monetary values (*monetary incentive* mode) and can also be translated into download rate (*bandwidth incentive* mode).
- *Hincent* is a max/min protocol making efficient utilization of network resources resulting in high throughput and lower transfer time.
- The prioritized rate allocation mechanism of *Hincent* allows some applications such as multi-view 3D streaming to assign higher rate to some flows (streams). The design has content lifetime feature to ensure efficient transmission of live and multi-view content.
- *Hincent* uses an efficient content index management scheme making it deployable in current networks without having to change the TCP/IP stack, routers or switches.
- We have presented an efficient algorithm which extends *Hincent* to use surrogate servers to help peers transfer contents faster.
- We have implemented *Hincent* in the NS2 simulator and evaluated its performance. Results show that it can achieve on average about 30% lower content transfer time when compared with existing schemes.
- We have experimented with *Hincent* using Apache SQL server, and have shown that *Hincent* scales.

The rest of the paper is organized as follows. In section II we present the *Hincent* protocol. In section III we present the methods *Hincent* uses to calculate the rates and prices which are used in the algorithms of the *Hincent* protocol. *Hincent* content source selection mechanism which is also used by the *Hincent* algorithms is presented in section IV. In section V we show how *Hincent* rate allocation is TCP friendly. Section VI discusses *Hincent* scenarios when a flow ends. A list of other server selection policies is presented in section VII. In section VIII we present the content index management component of Hincent. In section IX we show how Hincent content index management scales with the growth of the number of content records. Section X shows how our scheme deals with scarce backbone bandwidth. In section XI we show how *Hincent* can be extended using surrogate servers to help peers exchange contents. We evaluate the performance of *Hincent* in section XII. Analysis of related work is given in section XIII. Finally, we give conclusion of the paper in section XIV.

II. *Hincent* PROTOCOL

The *Hincent* protocol consists of network and content models, logical and physical architectures and algorithms described below.

A. Network and Content Model

The network model of *Hincent* consists of a graph $G = (N, E)$ of nodes N and edges E as shown in figure 1. The node set V consists of the CDN servers which provide content and the peers which provide and/or request for content. The edge set E consists of all edges going to and from the nodes.

All nodes are linked with each other over the Internet which may consist of multiple backbone networks. Each node has link with specified upload and download capacities it buys (gets) from network operators. The operator backbone network usually has enough bandwidth to provide bandwidth guarantee to the users (nodes). This is usually done using protocols such as the OSPFv3 as a Provider Edge to Customer Edge (PE-CE) Routing Protocol [14].

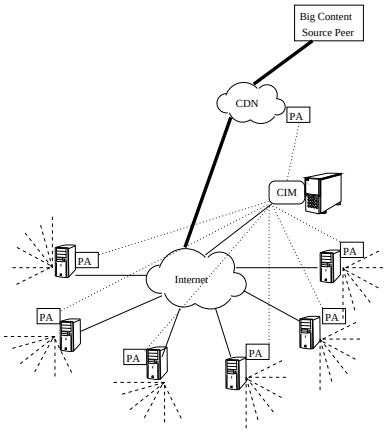


Fig. 1. The *Hincent* Architecture

The *Hincent* data model consists of content which is sent from the CDN servers or from some peers and exchanged between the peer nodes. We classify the data (contents) into none real-time ordinary static file (OSC), a realtime (live and none-live) streaming video content like 2-dimensional (2D) YouTube or a 3-dimensional (3D) video content [15]. The 3D Tele-Imersive content involves multiple streams from different view angles which have to be synchronized by the receiving end to produce a 3D multi-view streaming video. To synchronize the contents, *Hincent* uses content lifetime threshold based on how long a receiving node can buffer. For a stringent 3D Tele-Imersive environment, where the peers have to produce interactive content, the content lifetime becomes very small to ensure a very small delay. For most cases where nodes view the 3D content, the content life time can be relaxed.

B. Logical and Physical Architectures

The *Hincent* architecture aims to efficiently distribute content to network peers benefiting all content actors (content providers, content users and network operators). As shown in figure 2, it consists of a content information (index) manager (CIM) and peer agent (PA). A PA connects a peer with the CIM. A CIM registers peers and chooses content source to requesting peers. The CIM is made up of the light weight front end server (FES), content information database (CID), the complaint manager (CM) and the archive manager (AM). The CID consists of a database of contents information such as the source peers, source upload rates. The CM manages reports about misbehaving peers. The AM manages old content information and transaction logs to perform offline content index analysis. The FES forwards requests to register a new peer, a new content, or requests for a content, to the respective CID tables. The FES also forwards peer complaints to the CM. The CM contacts the AM for complaint history. The CIM archives old content state information at the AM.

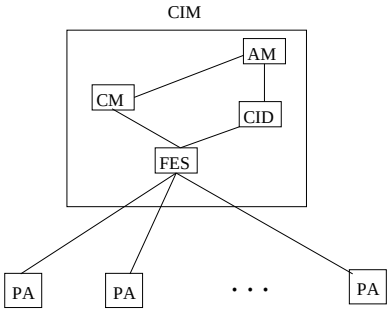


Fig. 2. The *Hincent* Logical Architecture

The *Hincent* physical architecture can be described by figure 1. The architecture consists of the peer nodes with their PA, the CIM and a big content source peer connected to its CDN with a bigger link. The big content source sends its content to the content distribution network (CDN) which informs the CIM of the new content. The content source which can be any peer with a PA can also inform the CIM of its content directly. The other peer nodes can then send a content request to the content information manager (CIM) via their peer agent (PA). The peers can get the content either from the CDN or other peers whichever gives the highest throughput to price ratio as discussed in the next section IV.

Hence *Hincent* consists of 3 main logical parts namely *content index manager* (CIM), *prioritized max/min rate allocation* (PRA) and *bandwidth and content pricing* (BCP) as shown in figure 3. These components interact with each other. The CIM consists of databases with information of peers and data contents. The PRA component is done with the help of the CIM and distributed peer agents (PA). It is where prioritized rate is calculated for each upload and download link of the peers and other main content servers. The rates are then used to choose a content source and to set the sending rates of the corresponding flows. BCP which is also done by the CIM and PA is a component where the bandwidth and content prices are calculated adaptively to ensure incentives between the participating peers. Peers which upload more, earn more credit which can be of monetary value or in terms of download bandwidth or content discounting.

We next discuss the *Hincent* algorithm involving the CIM and PA.

C. Hincent Algorithms

The *Hincent* algorithms are cooperatively run by the CIM and PA, to compute transmission rate and price (bandwidth, content) metrics for the content distribution. To obtain the rates at which each content is transmitted from one node to another node and the bandwidth usage price, *Hincent* first carries out temporary rate and price calculations at the CIM at every

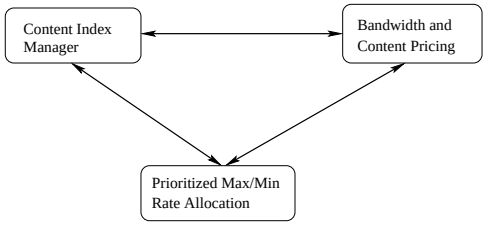


Fig. 3. *Hincent* Logical Components

request or at every control interval τ. The rates and prices are then sent to the PAs, updated by the PA and sent back to the CIM. The CIM then uses these rate and price values to select a content source (peer or CDN server) and determine the rate at which content is transmitted.

To define the *Hincent* rate and price metrics, we first present the following notations in table I. For each *Hincent* parameter $X \in \{R, C, Q, \hat{N}, N, n^j, R^j, M^j, p, \wp^j\}$, with j being a flow index, described in the table we use the notation,

$$X_{d,u} = \begin{cases} X_d & \text{if } X \text{ is a downlink } \textit{Hincent} \text{ parameter,} \\ X_u & \text{if } X \text{ is a uplink } \textit{Hincent} \text{ parameter.} \end{cases} \quad (1)$$

TABLE I
Hincent PARAMETERS

Variables	Description
$C_{d,u}$	Link capacity
τ	Control interval
$R_{d,u}(t)$	Base link rate allocation of the current interval (round)
$N_{d,u}(t)$	Number of flows in the link during the current round
$R^j_{d,u}(t)$	Link rate allocation of flow j for the current round
$M^j_{d,u}$	Minimum rate requirement of content flow j
$p_{d,u}(t)$	Per packet price
$\wp^j_{d,u}$	Priority weight of flow (stream or chunk) j

With the above notations, the *Hincent* algorithm consists of the following steps.

Initialization steps:

- In the *Hincent* deployment scenario each peer sets up a personal web (content) server (with the help of *Hincent*). The web-server can be hosted at a home server, a friend server, an ISP or a cloud.
- Each participating peer and CDN server first initialize their up link and down link base rates to the uplink and downlink capacities, they dedicate to the *Hincent* system.
- Each participating peer and CDN server also initialize their unit per packet price (bandwidth) to some value. In this study, the CIM sets the initial per packet bandwidth prices of the peers to a small fraction of real CDN bandwidth prices used by the Amazon CLoudFront [16]. Even though we consider only bandwidth price in this paper, the price may include other costs such as peer storage, energy, processing, content cost and other costs.
- Each participating peer and CDN server with a content then send these rate and price values along with other peer and content fields such as peer ID and content ID to the CIM.
- CIM authenticates and registers the requesting peers and the content sources.

Content request steps:

- Peer which is interested in a specific content sends (via its PA) a content request along with minimum rate requirement, M^i_u to the CIM. The most popular content information can be displayed by the CIM for other peers to see. Peers can also lookup the content from the CIM tables (via a web interface).
- If no peer has the desired content, the CIM sends the IP address of a CDN (cloud) server which has the content to the requesting peer and the IP address of the requesting peer to the selected CDN server. The CIM can also use existing search engines such as Bing and Google to look for the requested content. Once a requesting peer finds and clicks at the requested content, the index of the content can be stored as being available in the requesting peer by the CIM. Next time other peers request the CIM for the same content, the content can be directly served from the peer which got the content from the search engines. It is important that the CIM and the PA save the link to the original source of the content. This helps the PA to update the content and attract more customers with up-to-date content. Additional *Hincent* content servers can also keep a copy of the searched content and its original link to provide fresh content to peers and to monitor if the content source peers are offering fresh content. Peers have incentive to maintain fresh content as doing so attracts more customers (other peers).
- If there is (are) other peers which have a content requested by another peer, the CIM chooses the node (peer or CDN server) which gives the best metric (low price, high throughput) based on the content source selection policies discussed in section IV.

After the content request is received by the CIM, CIM and PA update steps are carried out before content transfer to avoid resource congestion and to achieve max/min resource (link) usage respectively.

CIM update steps:

- To reserve a minimum bandwidth requirement for the requesting peer, CIM subtracts M^i_u of request i from the remaining uplink capacity of the content source and M^i_d from the remaining downlink capacity of destination peer. This involves only a single subtraction operation. This remaining capacity is used in equation 5 of the rate calculation. If either of the remaining bandwidths is negative, the CIM informs the requesting peer that its request cannot be fulfilled.
- CIM increments the flow priority weight sum to be used in equation 5. This involves one addition instruction. The flow priorities are globally known to the CIM or specified by each requesting peer. The PA and the CIM then calculate the corresponding weights of the priorities.
- After accumulating the remaining bandwidth values and the sum of the priorities used in equation 5, the calcula-

tions of the base rate using equation 5 and price values using equation 7 can be done periodically to further reduce more computational overhead.
- CIM sends the IP of the selected content source along with the base upload rate $R_u(t)$ and the *contentHash* of the requested content to the requesting peer. The *contentHash* is to check for content integrity.
- CIM sends the base download rate $R_d(t)$ of the requesting node to the selected source.

When a PA of the content source and destination receive the rates $R_{d,u}$ of their uplink and downlink flows from its CIM, they performs the following.

PA update steps:

- Use the uplink and downlink rate values of each of the flows of its node received from its CIM to obtain the effective flow count for all uplink and downlink flows of its node using equations 9 and 8.
- Calculate new rate values using the effective flow count as given by equation 10. This new rate ensures that a capacity unused by some flows is being used by other flows making *Hincent* a max-min fair algorithm. This is because some uplink flows may be bottlenecked at the downlink and vice-versa.
- Calculate the new price value based on the new rate values using equation 7.
- Send the new base rate values obtained using equation 10 back to the CIM. The new price values can also be sent to the CIM saving the CIM some computational costs.
  The CIM then calculates its new price values and uses both the new rate and price values to select content sources (peers or CDN servers) for each request for content.

Rate enforcing and content download steps:

- Both content source and destination calculate the new rate $R^i_{d,u}$ values of each of their uplink and downlink flows (streams) i using equation 11.
- Both content source and destination enforce the rate allocation as follows. First the destination node sets its receive window w^i_r of flow i as

$$w^i_r = R^i_d(t)RTT^i. \tag{2}$$

  Then the corresponding source of the flow (stream) i sets its congestion window w_i as

$$w_i = min(w^i_r, R^i_u(t)RTT^i). \tag{3}$$

  If the bottleneck link is somewhere in the Internet which is described as "Internet" node in figure 1, then the destination of flow i sets its receive window size as given by equation 2. And the source of flow i sets its maximum congestion window size w^i_M as

$$w^i_M = R^i_u(t)RTT^i. \tag{4}$$

  Such a backbone bottleneck scenario can be detected by multiple packet losses after *Hincent* allocation, though we do not expect such a scenario to happen as discussed in section II-A. More on this will be discussed in section X.
- Requesting peer downloads the content from the source whose IP address it got from the CIM.

Price enforcing steps:

- Requesting peer via its PA asks for the *contentOldKey* from the CIM (CID) to decrypt the content it downloaded.
- The CIM increases the total amount $\ddot{E}$ of credit, the content source earns, and the total amount $\ddot{P}$, the receiving peer pays, each by the $contentSize \times p_{d,u}(t)$. *contentSize* is in packets.
- The CIM charges the requesting peer the specified amount and checks if the peer's balance has not fallen negative.
- If the requesting peer has enough credit (has paid for the content download), the CIM sends the *contentOldKey* to it (the peer). Otherwise the peer cannot decrypt the content after wasting its bandwidth.
- If the peer gets the decryption key, the CIM records the *contentID* of the downloaded content as available at the requesting peer unless the peer indicates it does not want to share the content. The efficient incentive mechanism of our protocol encourages peers to share contents.
- At the CIM when the flow of the requesting peer finishes (downloading the content), the remaining uplink bandwidth of the content source and the remaining downlink bandwidth of the receiving peer are increased by the minimum rate requirement of the flow which finished and the respective priority weights sums decrease by the priority weight of the flow which finished. CIM then updates the rates and prices using equations 5 and 7.

We next show how the *Hincent* rate and price are calculated.

III. *Hincent* Rate and Price Calculation

The temporary down-link (d) and up-link (u) rates of every node (peer or CDN server) are calculated by the CIM as

$$R_{d,u}(t) = \frac{C_{d,u} - \sum_j^{N_{d,u}} M^j_{d,u}}{\sum_j^{N_{d,u}} \wp^j_{d,u}} \tag{5}$$

where the notations are described in table I and $\wp^j_{d,u}$ is the priority weight of request j. If all requests have the same priority, $\sum_j^{N_{d,u}} \wp^j_{d,u} = N_{d,u}$.

The temporary uplink and downlink rates R^i_u and R^i_d of flow i are given by

$$R^i_{d,u} = M^i_{d,u} + \wp^i R_{d,u}(t). \tag{6}$$

The temporary per packet prices for the uplink (u) and downlink (d) are calculated as

$$p_{d,u}(t) = \frac{p_{d,u}(t-d) \times R_{d,u}(t-\tau)}{R_{d,u}(t)} \tag{7}$$

where the notations are also described in table I.

When a request for content is made, the temporary rate and price calculations ensure that the CIM does not result

in assigning requests to peers they do not have enough resources for. CIM leaves the refined distributed rate and price calculations to the peers.

With the temporary uplink rate of a flow k from a content source as R_u^k and the temporary downlink rate of the flow to the destination by R_d^k both obtained using equation 6, if $R_u^k > R_d^k$, then the content source of flow k should not send at the rate of R_u^k for flow k as it is bottlenecked in the last link to the destination. On the other hand if $R_u^k < R_d^k$, the destination node cannot receive (download) at the rate of R_d^k for the flow k. In these cases, other flows sharing the links with flow k should be able to use the corresponding uplink or downlink bandwidth unused by flow k to ensure that *Hincent* is max-min fair. To do this, some flows which cannot use the bandwidth allocated to them are counted as partial flows or fraction of a flow. We call such a count of a flow *an effective flow count*. The effective flow count of flow k at the source node is given by

$$n_u^k = \begin{cases} \frac{R_u^k}{R_d^k} & \text{if } R_d^k > R_u^k, \\ 1 & \text{otherwise.} \end{cases} \tag{8}$$

The effective flow count of flow k at the destination node is given by

$$n_d^k = \begin{cases} \frac{R_d^k}{R_u^k} & \text{if } R_u^k > R_d^k, \\ 1 & \text{otherwise.} \end{cases} \tag{9}$$

Each PA then obtains new uplink and downlink base rate values as

$$R_{d,u}(t) = \frac{C_{d,u} - \sum_j^{N_{d,u}} M_{d,u}^j}{\sum_j^{N_{d,u}} \wp_{d,u}^j n_{d,u}^j}. \tag{10}$$

The new per packet prices for the uplink and downlink of a node are then obtained using equation 7.

Besides, a node resets the up and downlink rates of each of its' flow i as

$$R_{d,u}^i = M_{d,u}^i + n_{d,u}^i \wp^i R_{d,u}(t). \tag{11}$$

Equivalently, the uplink rate R_u^i of the flow i at a node can also be calculated as

$$R_u^i = M_u^i + n_u^i \wp^i R_u(t). \tag{12}$$

So far we have considered the *monetary incentive* mode of *Hincent*. The monetary incentive can also be converted to a upload *bandwidth incentive* using the ratio of the total amount to pay to the total credit earned. To do this, the CIM informs the content source to rate-limit the requesting peer at a base rate of

$$\ddot{R} = \ddot{w}(\ddot{E}, \ddot{P}) \times min(R_d(t), R_u(t))$$

where $\ddot{w}(\ddot{E}, \ddot{P})$ is the weight function of the total monetary amount $\ddot{E}$ the requesting peer has earned and the total amount $\ddot{P}$ the peer has to pay. The *min* is a minimum function. In this study we set

$$\ddot{w}(\ddot{E}, \ddot{P}) = \frac{\ddot{E}}{\ddot{P}}. \tag{13}$$

Other pricing and weight functions can also be used in *Hincent*. The new weights $\tilde{\wp}_u^j$ of every request j from the requesting peer is then set as $\tilde{\wp}_u^j = \wp_u^j \ddot{w}(\ddot{E}, \ddot{P})$. This new weight is the product of the peer weight, $\ddot{w}(\ddot{E}, \ddot{P})$, and the flow (stream) priority weight, $\wp_u^j$. CIM obtains the rate allocation of the request j made by the peer as $\ddot{R}^j = M_u^j + \tilde{\wp}_u^j \ddot{R}$.

IV. Content Source Selection

Once the CIM receives the new rate values from each PA, it obtains the new price values using equation 7. Then a content source for the requesting peer is selected based on the policy discussed below.

A. Highest Rate to Price Ratio Policy (HRPR)

In this HRPR policy, the CIM keeps the ratio

$$K_{d,u}(t) = R_{d,u}(t)/p_{d,u}(t) \tag{14}$$

of the rates to their respective prices in its peer table. When a node requests for a content, the CIM chooses a content source which gives the highest value of $K_{d,u}(t)$. This approach enables the CIM to choose a node which gives the highest rate with the lowest price. This policy takes locality into account, serving requests using local sources which give the HRPR. It can also be applied to social groups, selecting the best (with HRPR) content sources in the group for requesting peers.

V. *Hincent* is TCP friendly

In this section we discuss how *Hincent* deals with TCP friendliness.

Theorem 1: A *Hincent* rate allocation of a flow which is not bottlenecked at a link l is TCP friendly to all flows sharing link l.

Proof: If a flow i is not bottlenecked at link l, it cannot congest link l regardless of how much its sending rate increases. This is because the flow i has another bottleneck which limits its sending rate. This in turn means that TCP flows sharing link l with flow i have enough bandwidth at link l to use. This implies that TCP fairness is not an issue at link l and flow i is TCP friendly. ■

Even though *Hincent* handles scenarios where the bottleneck link can be somewhere in the backbone network, the bottleneck link in the *Hincent* architecture is usually going to be at the last mile links to and from the peers. This is because (1) users (peers) usually buy a guaranteed bandwidth and (2) the peers which can use a specific peer as a source of their content are usually scattered over a wide area each using different paths in the backbone network. Hence, if the *Hincent* flows are not bottlenecked at a link which they share with TCP, then they are TCP friendly based on the above theorem 1. In a scenario where the bottleneck link is in the backbone network, the *Hincent* flows will drop or delay packets. This congestion signal can be detected by the PA of each peer which

counts the number of successfully transmitted packets. The PA compares this count over a time interval against the minimum of the uplink and downlink rates of the flow. If the PA finds that the backbone network is congested, it uses the maximum congestion window and receive window to enforce the rate allocations as discussed in section X.

VI. When a Flow Ends

When a peer wants to end a flow (stream) due to for instance 3D view change, the node sends the contentID of the flow (stream) it needs to end. The CIM then finds the corresponding global contentID in its content table, removes the contentID and releases the associated resources. It then updates the corresponding content source and destination rate and price values. The CIM also finds a new content source to all other peers which are actively downloading the content from the peer which wants to end it. Here, the CIM uses the original content source as the new content source for the peers which are using the content whose source is ending it. This is because if a new peer (which is not the original content source) is chosen to be the new source of the content, it is difficult to find (trace) out whether one of the parents (ancestors) of this chosen peer is the peer which is ending the content or not.

VII. Other Server Selection Policies

In this section we discuss server selection policies other than the HRPR policy discussed in section IV-A.

A. Highest Rate Policy

In this policy the CIM selects a content source which provides the highest rate to each node irrespective of the price. So a node which is allowed to download from a content source with the highest rate pays the corresponding price. The nodes can also earn credit by allowing other nodes to download from them and then get a service whose price is equivalent to the credit they have earned as discussed in section III.

B. Best Rate Fit Policy

When a node requests the CIM for a content, the CIM can also choose a content source whose upload rate is the smallest value greater than the download rate of the requesting node. This approach allows the CIM to do a best fit allocation to allow big upload requests.

C. Smallest Price Policy

If a node which requests for a content doesn't want to pay more or doesn't want to spend more of its credit, it can request a smallest price policy. In this case, the CIM chooses a content source with an upload value of at least as much as the minimum required rate for the content and with the smallest price.

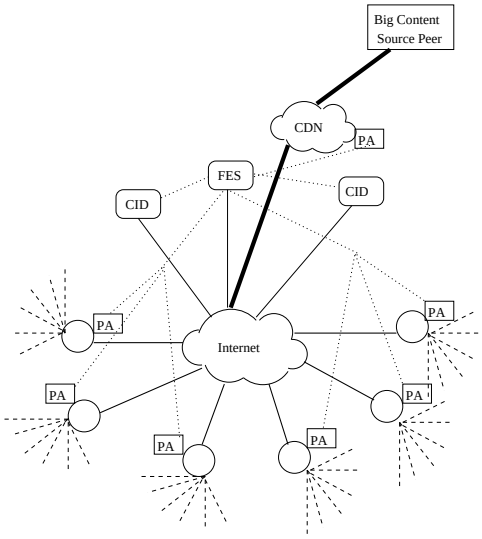


Fig. 4. The *Hincent* Architecture

D. Lowest Latency (Local Network) Policy

A user's request may have some latency constraints. In this case a user may request a node with the shortest latency. To deal with this scenario, we group peers with similar IP prefixes together. This can be done by hashing the most-significant bit-group in the IP address of the content request packets of the registering peers. We can then have one CIM responsible for each group of users (peer nodes) forming a hierarchical structure of content information managers as shown in figure 4. This policy can have a significant advantage in reducing backbone network link congestion as many requests can be served locally. This is another benefit to network operators. Besides, users in the same geographical location may tend to have interest to the same content making it easy for the content source selection algorithm to decide.

To use this policy, users send a request to the FES of the CIM which then hashes the requester's IP prefix values and forwards them to their respective CID tables. This approach also allows *Hincent* to scale as discussed in section IX.

E. Small Content Lifetime (Hop Count) Policy

The peers in the *Hincent* have a strong incentive to store and share the contents they download. As every upload can result in credit which can translate to monetary rewards or high download rate. A node can also inform the content index manager (CIM) that it does not want to serve a specific content. Besides, a peer which has big enough buffer can store early arriving streams to create a 3D tele-Imersive [10] view along with other streams which arrive late. In a scenario where a significant number of peers have limited buffer, *Hincent* can follow a *small content lifetime* policy. In this policy, the CIM uses a content hop count field in its content index database (CID) along with locality information. Here a content source with the lowest hop count is selected to serve the requesting peers. The CIM first tries to find such content in the local CID. If the content with the desired hop count cannot be found in the

local CID, it is searched in the master *tblSelectedSource* table as shown in section IX-A. If such content with the desired hop count cannot be found, the default *highest rate to price ratio* policy discussed in section IV-A is used.

F. Private Group Policy

This *Hincent* policy allows content to be shared within a specific group of peers which can be social or organizational groups. Each private group can form its own CIM with any of the above server selection policies. This can enable *Hincent* to deploy Facebook like applications such as the Diaspora [17], [18]. A distributed network of CIMs can also be formed where CIMs exchange public content information based on privacy settings in an adhoc or hierarchical manner. Any peer can then subscribe to different CIMs for different contents forming a distributed content networking.

So far we have been discussing the two major components of *Hincent* which deal with the prioritized rate allocation and resource pricing. After presenting mechanisms of how the uplink and downlink rates for each peer and the corresponding bandwidth prices are calculated by the CIM and PA we have also discussed how the CIM uses these metrics to select a content source for a requesting peer. We next present an efficient content index management scheme which the CIM uses to select the best content source for a requesting peer.

VIII. Content Index Management

In *Hincent*, some peers or content providers provide content by registering their content information at the content index manager (CIM). Other peers request the CIM for a specific content. In this section we show how such contents are registered, requested and their source selected.

A. Content Index Database

The registered content information is stored at the content index database (CID) which is part of the CIM system as shown in figure 2. The CID consists of the *tblPeer*, *tblContent*, *tblSelectedSource*, *tblRequestedContent* tables as shown in figure 5. The *tblPeerContent* table is used to link the *tblPeer* and *tblContent* in a many-to-many relationship.

1) Peer Table: The *tblPeer* contains the fields described in table II. Initial content providers need to fill in all the fields of this table. The *peerInfo* contains real content provider information such as telephone number, address and/or credit card number. Such confirmed information holds each content provider accountable for the nature of the content provided. The *peerInfo* field is also used by the content providers to charge peers for none-free contents. Once a peer receives a content, the CID registers the peer as having the content unless the peer indicates that it does not want to serve the content. The peers which are not the original sources of the content do not have to provide their *peerInfo* unless they want to receive monetary value of the credit they earn. The *peerID* field is the primary key of the *tblPeer*. It is preferred to be the IP address of the peer. The peers have incentives to provide their correct IP addresses. This is because if a peer gives a wrong IP address, it can not get a content as a source sends its content to an IP address it obtains from the CIM.

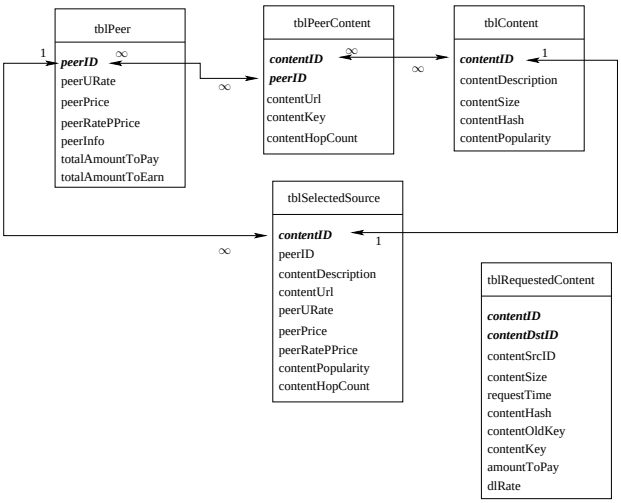


Fig. 5. The CID Architecture

TABLE II
Peer Table Fields

Field name	Description
peerID	Unique peer identifier
peerURate	Current base uplink rate, $R_u(t)$ of a peer calculated using equation 5
peerPrice	Per unit uplink cost of a peer node
peerRatePPrice	Peer rate per price calculated using equation 14
peerInfo	Real content provider information
totalAmountToPay	Total monetary amount a peer needs to pay for downloading a content
totalAmountToEarn	Total monetary amount a peer earns for uploading a content

If the peer is just joining the CIM, its uplink rate, *peerURate*, is the the total uplink capacity it uses to earn credit from other peers to which it uploads content. The peer has an incentive to dedicate more uplink capacity, as more uplink capacity can bring the peer more credit (monetary values). After the initial calculation by the CIM using equation 5, *peerURate* is updated by each peer using equation 10. To minimize the computation load of the CID, the *peerURate* can also be entirely calculated by the peers in a distributed manner and sent to the CIM every control interval τ. If the peers send their download rates to the CID and if there is enough server processing (computation) capacity at the CIM, all rate computations given by equations 5 and 10 can also be done by the CIM servers in a centralized manner. In this paper we use the approach where initial simple rate computation is done by the CIM servers and the more detailed rate update computation is done by the peers. The peers then send the update to the CIM servers.

The *peerPrice* in our study is per packet cost where one packet in this study is 1000 Bytes. The *peerPrice* is initially set to be the unit content cost plus basic initial user defined link cost. The content cost is zero for a free content scenario and the initial link cost in our study is determined by the CIM system. After the initial cost, *peerPrice* is calculated adaptively

by the CID servers using equation 7 for each link.

The default content source selection policy we use in this study is the *Highest rate to Price Ratio Policy* discussed in section IV-A. To implement this policy the *tblPeer* maintains the peer rate per price *peerRatePPrice* field.

The *amountToPay* and *amountToEarn* fields are updated by the *tblRequestedContent* table. A peer gets an additional amount in dollars for each content it serves and pays a certain amount for each content it downloads.

2) Content Information Table: The second table the CID keeps is the content information table which we call *tblContent* in this paper. This table contains the fields shown in table III.

TABLE III
CONTENT INFORMATION TABLE FIELDS

Field name	Description
contentID	Uniquely identifies content chunk or stream in the CIM
contentDescription	A textual description of the content
contentSize	Size of the content in KB
contentHash	To check for content integrity
contentPopularity	The number of times a content with *contentID* is requested

The *contentSize* is used by the CIM to charge the peer which receives the content. The CID uses this content size to obtain the per content amount a peer has to pay. The *contentHash* is used by the content receiving peer to check for content integrity. Every time a content is selected by a peer, the popularity of the content increases.

3) Peer-Content Linking Table: This *tblPeerContent* links the *tblPeer* with the *tblContent* in a many-to-many relationship. To achieve this, *tblPeerContent* consists of the primary keys *peerID* and *contentID* of *tblPeer* and *tblContent* tables respectively. The table also contains the peer specific fields, *contentUrl*, *contentKey* and *contentHopCount*. The current location of the content in a peer with *peerID* is *contentUrl*. The source peer encrypts its content with the symmetric key *contentKey*. After a peer receives a content from another peer or from a the original content server, it requests the CID (*tblRequestedContent*) for the key to decrypt the content. The *contentHopCount* is set to 1 if the peer is the original content source. Every other peer which receives the content increments the value of the field by 1. This field along with locality information for instance helps estimate the streaming content age since its initial distribution.

4) Selected Source Table: From all the original content servers and peers which have a specific content, a source for a requested content is selected based on the content source selection policy discussed in section IV above. For each content source selection policy, a table called *tblSelectedSource* is produced by a query from the *tblPeer*, *tblPeerContent* and *tblContent* tables. For the *Highest Rate to Price Ratio Policy* (HRPR) used in this paper, the *tblSelectedSource* has the fields, *contentID*, *peerID*, *contentDescription*, *contentUrl*, *peerURate*, *peerPrice peerRatePPrice* and *contentPopularity*. This table can be sorted in descending order of popularity to put the most popular contents at the top even though every content can be looked up in constant time.

5) Requested Content table: The requested content table, *tblRequestedContent*, consists of the fields, *contentID*, *contentSize*, *contentSrcID*, *contentDstID*, *requestTime*, *contenOldtKey*, *contentKey*, *amountToPay* and *dlRate*. The *contentID* and *contentSize* fields correspond to the the requested content. The *contentSrcID* field is the *peerID* of the peer or server which is selected to serve the content. The *contentDstID* field is the *peerID* of the content requesting peer. The field, *requesTime* is the time when a request for the specific content was made. The *contenOldtKey* field is a symmetric key with which the content was encrypted and by which the content receiver will decrypt the content. Once a peer with *contentDstID* requests for this key to decrypt the content it downloaded, its *amountToPay* value is set to the product of the *contentSize* and the per packet price, *peerPrice*, of the source link. The *totalAmountToPay* of the peer with *contentDstID* and the *totalAmountToEarn* of the peer or server with *contentSrcID* that serves the content each increase by *amountToPay*. The *contentKey* field is a new symmetric key generated by the CID for the content downloaded by the peer. The content requesting peer uses this key to encrypt the content when selected by the CIM to serve the content. Once the *contenOldtKey* is successfully received by the peer which requested the content, and after other tables of *contentSrcID* and *contentDstID* are updated, the record entry of these fields in *tblRequestedContent* is deleted. The *dlRate* field is set to the minimum of the downlink (to the destination) and uplink (from the source) rates of the requested content.

In the next section we discuss how the CID tables scale with the growth in the number of content and peer record entities.

IX. SCALING USER AND TRANSACTION MANAGEMENT

In this section we discuss how *Hincent* scales to an increase in the number of users and with the multiple variations in the request arrival and completion patterns. The CID tables can be scaled with increasing number of peers and contents by using multiple data center like servers along with appropriate hash functions. If the number of servers available for the *tblPeer* table is S_p, $sigBits(peerID)$ gives the integral value corresponding to the most significant bits of the peerID field. How many significant bits of the peerIDs we take depends on how many content entries we have. Taking fewer significant bits for instance means we need fewer servers (smaller S_p) as more peerIDs can be mapped to a single server. A record for *peerID* goes to *tblPeer* located at server $sigBits(peerID) \ mod \ S_p$. Here the servers are identified by positive integral values and mod is the modulo operation. A record for *contentID* of *peerID* goes to *tblContent* located at server $sigBits(peerID) \ mod \ S_p$. This ensures that the content and peer information are located in the same server for easier local look-up.

Such hashing by *sigBits(peerID)* helps that content information of peers whose IP addresses have the same domain go to the same server. In this case if a peer in one index server is selected by the CID as a source of a content to another peer in the same index server, then the content source selection

strategy becomes local. Such local content source selection mechanism can help peers achieve low download latency as the content can be served from another peer in their local network.

When the request arrival and completion vary so much, the PA needs to recompute the rate given by equation 10 multiple times. Furthermore the PA needs to update the rate values at the respective CIM. Since each CIM obtains temporary rates using equation 5, the PA does not have to send every update to the CIM. The PA can send updates every user-defined control intervals.

Equation 5 used by the CIM only needs one subtraction (addition) and one division per new flow request arrival or departure to obtain a temporary uplink rate for each peer. It also needs one multiplication and one division to obtain the temporary price given by equation 14. Since the process is adaptive, some CIM rate and price updates can as well be skipped as they can be updated by the rate the PA send for each of their links. The CIMs using equation 5 also do not need to obtain the temporary downlink rates and prices. The downlink rates and prices can be sent by the content requesting peer. In these cases the CIM only needs to check if the selected source has enough remaining upload link capacity to satisfy a minimum rate requirement $M_{d,u}^{j}$ of the request j.

A. Database Partition and Aggregate

Assigning *tblPeer* and *tblContent* tables to different index servers based on the peer ID (IP) essentially partitions the CID into multiple local databases. Each local database matches content source and destination peers located in the same network domain and local area. We call such content matching a *local content source selection strategy*.

If content cannot be found in a local network or if peers in other local networks can provide a higher upload rate and lower price, then the source selected to serve a content can be from a different network domain, different area or even different country. Such a content source selection strategy where a content source can be chosen from a different network domain (area) is called *global content source selection*.

To achieve global content source selection, the CID needs to know a source with the highest upload rate and lowest per packet price for the requested content. The CIM achieves this by using a map/reduce [19] like framework as shown in figure 6. For the content source selection strategy we use in this paper, each local CID database's *tblPeerContent* is sorted in descending orders by *contentPopularity* and then *peerRatePPrice* for each content. So here we have each *tblPeerContent* information *mapped* to many local index servers (many CIDs).

For each content, a record with the highest *peerRatePPrice* among all the *tblContent* tables in each local CID database is selected *(reduced)* into the *tblSelectedSource* table and placed in another index server. This is like the reduction phase in the map/reduce framework where the *maximum of* is the reduce function. Each CID continuously sorts the *tblPeerContent* table by the *peerRatePPrice* field for each content with the changes in the upload rates and prices of the corresponding peer.

If the value of *peerRatePPrice* field in a *tblPeer* table changes, first, each *contentID* content of the *peerID* peer in the *tblPeerContent* is sorted in descending order of *peerRatePPrice*. Then for each content of *peerID* in the *tblSelectedSource* table, if the highest *peerRatePPrice* of *peerID* is higher than the *peerRatePPrice* of the corresponding content in *tblSelectedSource*, then the values of the *peerID* and *peerRatePPrice* fields in the *tblSelectedSource* table are replaced with the corresponding values in the *tblPeer*. The *tblSelectedSource* table is sorted by *peerID*. Hence all contents of the *peerID* field are located once the first content of *peerID* is found. Such a procedure of constantly updating the *tblSelectedSource* table ensures that the table always consists of the list of contents given by the peers with the highest upload rate and lowest price (highest rate to price ratio).

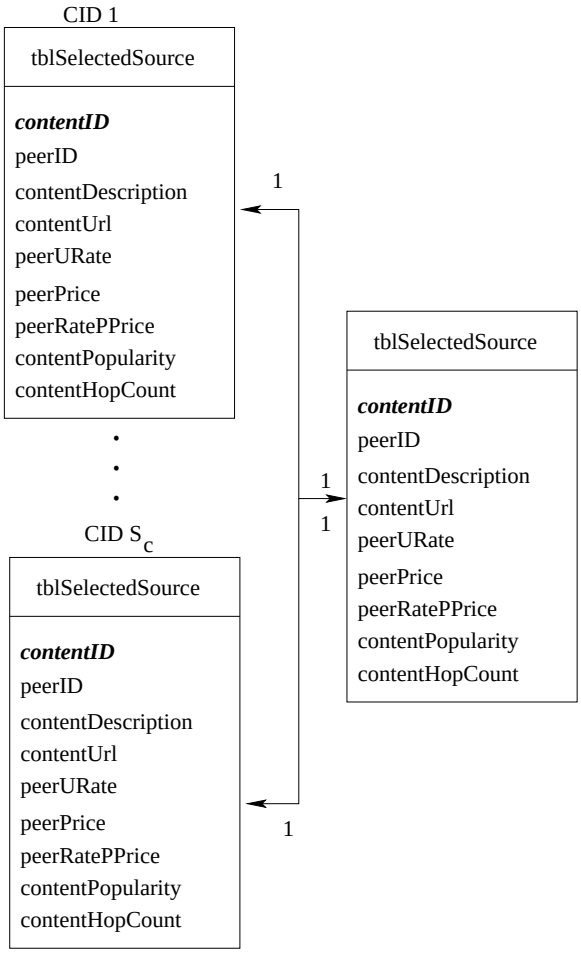


Fig. 6. The CID Partition and Aggregation

B. CID Complexity Analysis

The CID operation of adding new peers to the *tblPeer* is of constant order $\mathrm{O}(1)$ as the *tblPeer* does not have to be sorted out. Each content in the *tblPeerContent* has to be sorted out by *peerRatePPrice*. Hence adding a new content entry to the *tblPeerContent* table has a complexity of order $\mathrm{O}(log(N_c))$ where N_c is the number of peers which have the same content.

Whenever a peer gets a content that it requested, then (1) the uplink rate of the content source decreases according equation 5 and (2) the peer which gets the content becomes one of the content sources. These two operations require two $\mathrm{O}\big(log(N_c)\big)$ operations for each content. As the *tblPeerContent* in each of the CID partition is sorted, updating the *tblSelectedSource* for each of its contents is of constant order.

When *tblRequestedContent* is updated upon a successful download of a content by a peer, the corresponding values of *peerRatePPrice* and *totalAmountToEarn* of a source peer or server and the *totalAmountToPay* of a receiving peer are updated in constant time by using the matching *peerID* field.

X. *Hincent* WITH SCARCE BACKBONE BANDWIDTH

The backbone links in the Internet which the nodes use and which are represented by the "Internet" node in figure 1 are not usually congested as can also be seen from [20]. Each user of the *Hincent* mechanism can also have bandwidth service level agreement from the operators which guarantee the desired capacity. Under this scenario the only bottleneck links are the last links to and from the *Hincent* peer nodes. Hence the sources of the desired contents can set their congestion window sizes (*cwnd*) to the product of their uplink rate value calculated using equation 11 and their round trip time (RTT).

If the bottleneck link is somewhere in the Internet which is described as "Internet" node in figure 1, then the destination of flow i sets its receive window size as given by equation 2. And the source of flow i obtains its maximum congestion window size w_M^i using equation 4.

A node can detect whether or not the bottleneck is in the link other than the last links to and from the source and destination peers using different ways. For instance if a packet loss is observed for flow i after the rate is enforced using equation 3, then the TCP source of flow i can assume the bottleneck link is other than the last links to/from the source and destination nodes. The PA of the receiving end can also count the number of received packets (bytes) per unit time to obtain the actual download rate per content. Similarly the PA of the content source can also estimate its uplink rate of a specific content by counting the number of successfully acknowledged packets (bytes) per unit time. The PA of the source and destination of the content then report this rate to the CIM per a specific content. The CIM then replaces the *peerURate* of the content in *tblSelectedSource* with the minimum of these two values. The source and destination peer also update their rate calculations using equations 8, 9 and 10 to re-allocate unused capacities to other requests.

The *tblRequestedContent* of the CID has the *requestTime* and *contentSize* fields. When a peer receives a content, it requests the CID for the decryption key. The CID of the CIM can then compare the $actualTime = currentTime - requestTime$ against $promisedTime = contentSize\,/\,dlRate$, where $currentTime$ is the time when the request for the decryption key arrives at the CIM and $dlRate$ is the minimum of the uplink rate and downlink rate of the requested content. If the $actualTime > promisedTime + toleranceVal$, then the CIM via its CM concludes that the source is not uploading at the rate it suggested. Here $toleranceVal$ is a user-defined tolerance value. In cases where the requested content is a video stream, the source of the content can stream its frames by scheduling them at $\frac{1}{R_u^i}$ apart where R_u^i is given by equation 11.

XI. *Hincent* USING SURROGATE SERVERS

In the *Hincent* deployment scenario presented in the previous sections, each peer uses a personal web (content) server similar to the Diaspora social network [17], [18]. The personal web server can be hosted at a home server, at a friend server or at an ISP. An extension of *Hincent* can also be implemented in big content distribution services such as Google (YouTube) or other overlay (private) networks such as [15] using cloud/cloudlet surrogate servers geographically distributed in a wide area as described in figure 7. The servers are equipped with OpenFlow vSwitches (switches/routers). The links of the network shown in figure 7 can be dedicated tunnels or overlay links over the Internet. If the links are overlay, their capacity can be estimated using bandwidth estimators. We next discussed how *Hincent* distributes content using such surrogate servers for the peers uploading and requesting for content using OpenFlow vSwitchs [21], [22].

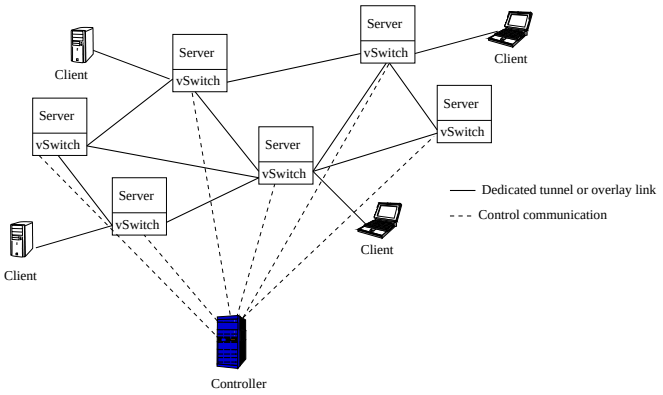


Fig. 7. *Hincent* with Surrogate Servers

As shown in figure 7, peer clients upload and get contents from their nearest surrogate servers using the following steps.

Content storage steps:

- A peer client which wants to share its content with other peers or which wants to store its data in some distributed servers, sends its request to a light weight FES (frontend server) associated with the controller shown in figure 7. The FES can also be associated with each surrogate server.
- The FES hashes the request and forwards it to the closest server using (first significant bits of) IP address matching. This ensures that a request is handled by surrogate server in the locality of the requesting peer.
- The FES also forwards the content information to a modified *tblContent* which resides in the CID (content index database) of the controller. The modified *tblContent*

includes the ID of the selected (by FES) surrogate server where the content initially resides in addition to the other fields described in table III.

- The controller informs all surrogate servers of the new content. The surrogate server which is initially selected to host the content can also share the contentID information with the other servers similar to the way link state information is shared.
- Each surrogate server adds a new record to its *tblServContent* table which has the fields *contentID*, *contentPopularity*, *sourceID* and *servURate*. The *contentID* and *sourceID* values are obtained from the controller or from the surrogate server which is selected to initially host the content. The value of *servURate* along with the path to the server with the content is obtained using a max/min routing algorithm described in [23] and [24]. The link metric of the network of surrogate servers described in figure 7 is a cross-layer (routing and congestion control) rate metric obtained using the schemes described in [24]. As discussed in [24], this rate metric can be obtained using vSwitch (OpenFlow) per flow packet counts (statefull) or using surrogate server assistance (stateless).
- Each surrogate server updates its *tblServContent* table sorting each contentID entry in decreasing order of *servURate* value every time route computation to other servers is done. The servers also sort the content entries in decreasing order of popularity. When a peer request for content is made, a selected server gets the content from another server with the highest *servURate* (for the requesting peer). Here *servURate* is the bottleneck update rate from a source surrogate server to the destination surrogate server.

Content retrieval steps:

- A peer client requests for a content by contacting the FES. The FES seamlessly hashes the client ID and forwards its request to a surrogate server which is the nearest (closest IP address for instance) to the requesting client.
- If the surrogate server has the content, it directly transmits it to the requesting client. Otherwise, the surrogate server looks up its *tblServContent* table for the best (highest *servURate*) other server with the requested content. It then starts a QCP [24] session (can also be TCP) to the selected (highest *servURate*) server and gets the requested content for the requesting peer.
- The server which downloaded the content on the behalf of the peer client stores (caches) the content and informs the controller CID that it also has the content. The controller CID then informs other servers that the server also has the content.
- This surrogate server which obtained the content from another server also transfers the content to the requesting peer. The connection between the peer client and its surrogate server can also be QCP if the peer has a dedicated tunnel connecting it with its server. Otherwise it can be a TCP connection.

We have performed detailed experimental analysis to evaluate the performance of *Hincent* as shown in the next sections.

XII. Evaluation

In this section we evaluate the performance of *Hincent* and all its components using simulation. We implemented *Hincent* in the NS2 simulation package. We also implemented the CID of *Hincent* using Apache SQL server [25]. After discussing the simulation setup, we present detailed trace-based packet level simulation experiments. We then show how *Hincent* content management scales using Apache SQL implementation experiments.

A. Simulation Setup

We use a simulation topology similar to the one given in figure 1. For the simulation the upload and download capacities of the links to and from the peers is 15Mbps. The link capacity to and from the CDN is $n_{peers} \times 15\,Mbps$, where n_{peers} is the number of peers. The propagation delay between the peers is taken from 4 hour PlanetLab traces [26]. The average CDN bandwidth price taken from the Amazon CloudFront [16] is $avg\_cdnPrice = \$0.176$ per GB of traffic. The initial peer bandwidth price is $avg\_cdnPrice/(2.0 \times n_{peers})$. This price adaptively increases as the peer rate decreases with more demands based on equation 7. We run different sets of experiments as shown in the following sections.

B. Pure CDN Vs Hincent-Based Schemes

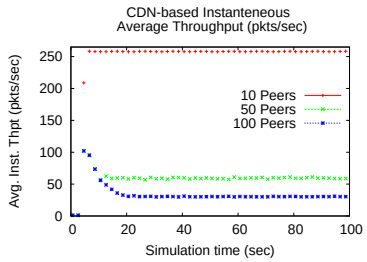
Figure 8 shows how the *Hincent*-based scheme scales with the growing number of content requesting peers when compared with the pure CDN-based approach. This result is consistent with detailed study [5] which shows that the hybrid CDN-P2P can significantly reduce the cost of content distribution bandwidth.

C. Other P2P schemes Vs Hincent

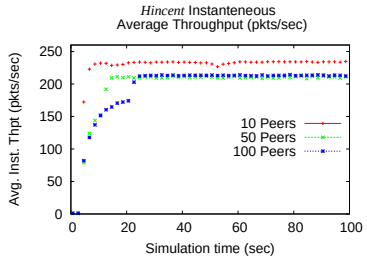
We have also compared the performance of *Hincent* against other hybrid P2P and CDN schemes in terms of average chunk completion time (ACCT). Previous hybrid P2P and CDN schemes such as the Dandelion [3], PACE [8] use TCP as their transport protocol. So we show how these schemes using TCP compare against *Hincent* by fixing the content source selection mechanism to be the same (based on *Hincent*) for both.

For this experiment we use 8 files with content $i, (1 \leq i \leq 8)$ having file size $500i$ KB and chunk size i is $50i$ KB. Inter-content chunk request time is 0.5 seconds. Contents are requested at the same time. Each file (content) is divided into equal chunks. Content popularity is 5 for each of the contents. For the TCP-based and the *Hincent* approaches content destination and source are the same. For these experiments we set the minimum flow rate to 0.0 and all chunks have the same priority levels.

Figure 9 shows that the ACCT and average maximum CCT (Max CCT) are much smaller in *Hincent* than the TCP-based approaches (PACE, Dandelion). The Max CCT is the content



(a) Pure CDN based Approach



(b) *Hincent* based Approach

Fig. 8. Pure CDN Versus *Hincent*-based Approach

(file) completion time as a file download is complete after its latest chunk is downloaded.

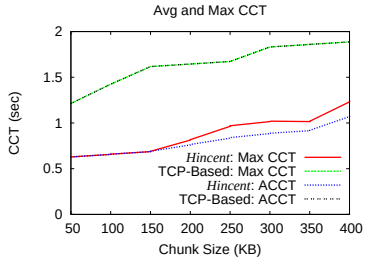


Fig. 9. Avg and Max CCT of *Hincent* Vs TCP-Based Approaches

D. 3D Streaming Result

For the 3D streaming experiments, we use a setup which emulates [15] with 6 streams. Each stream demands a minimum of 1Mbps capacity. Each stream i, $1 \leq i \leq 6$ has a priority weight of $1/i$. We used a content lifetime of 2.5 seconds for the streaming. So if a stream at a peer is older than 2.5 seconds, the CIM does not register the peer as having the content.

Figure 10 demonstrates the priority and minimum rate mechanisms of *Hincent*. As shown in the figure, stream 1 which has the highest priority weight gets highest throughput. The throughput of the other streams follows their priority weights.

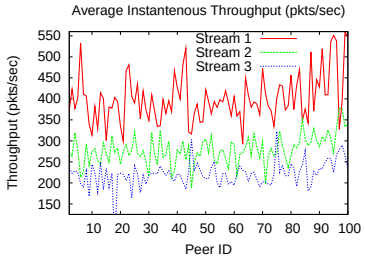


Fig. 10. Avg Instantaneous Throughput Per Peer for Streams 1, 2 and 3

Figure 11 also shows how the instantaneous throughput of the different streams evolve with time. All these plots show how efficiently *Hincent* enforces the priority based rate allocations. For readability and in the interest of space, plots with streams 4, 5 and 6 are omitted.

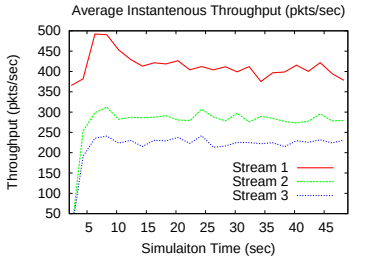


Fig. 11. Avg Instantaneous Throughput Over Time for Streams 1, 2 and 3

E. More Trace-Based Experiments

We have also conducted experiments based on the trace results presented in [27] for the content size distribution, [28] for the content popularity distribution and [29] for distribution of the flow arrival process. Since we could not obtain the raw trace data, we constructed the trace values (data points) from the plots given in these papers. We next present the trace extraction methodologies we used.

1) Extracting file size distribution: Based on the nature of the file size trace plot of US-Campus given in figure 4 of [27], we constructed piece-wise linear functions given by equation 15.

$$FS = \begin{cases} u(1,5), & cdf <= 0.17, \\ \frac{10-5}{0.18-0.17}(u(0,1)-0.17)+5.0, & 0.17 < cdf \leq 0.18, \\ \frac{200-10}{0.204-0.18}(u(0,1)-0.18)+10, & 0.18 < cdf \leq 0.204, \\ \frac{1000-200}{0.25-0.204}(u(0,1)-0.204)+200, & 0.204 < cdf \leq 0.25, \\ \frac{30000-1000}{0.96-0.25}(u(0,1)-0.25)+1000, & 0.25 < cdf \leq 0.96, \\ \frac{70000-30000}{0.99-0.96}(u(0,1)-0.96)+30000, & 0.96 < cdf \leq 0.99, \\ \frac{100000-70000}{1.0-0.99}(u(0,1)-0.99)+70000, & 0.99 < cdf \leq 1.0. \end{cases} \quad (15)$$

In equation 15, the function $u(a,b)$ generates a uniform random number between a and b and cdf is the CDF of the file size trace plot. In the simulation, we first generate a uniform random number between 0 and 1. We then obtain the file size (FS) value as a function of the generate value using equation 15.

2) Extracting content popularity distribution: A Gamma distribution curve with a shape parameter of $\tilde{k} = 0.372$ and a scale parameter of $\theta = 23910$ is fitted to Youtube video content popularity distribution traces in figure 7 of [28]. The content popularity distribution in the paper which refers to the number of views of videos considers about $N_V = 1.6 \times 10^5$ videos. We normalized the scale parameter θ of the distribution by the number N_V of distinct videos so as to use it with simulation studies involving a different number of videos. The normalization steps are as follows.

With n_v as the total number of distinct (unique) video flows to be simulated, and p_v the average popularity of the videos, $n_v p_v$ is the total number of videos to be simulated. With a simulation time of t_s seconds and average video request arrival rate of λ_s flows per second, we have

$$n_v = \frac{t_s \lambda_s}{p_v}. \tag{16}$$

To obtain p_v, we normalize the number N_V of traced videos by the mean $\tilde{k}\theta$ of the Gamma popularity distribution as

$$\frac{n_v}{p_v} = \frac{N_V}{\tilde{k}\theta}. \tag{17}$$

Combining equations 16 and 17, we get the popularity value as

$$p_v = \sqrt{\frac{\tilde{k}\theta t_s \lambda_s}{N_V}}. \tag{18}$$

Using equation 18 in equation 16 we also obtain the number of distinct videos in the simulation.

3) Flow arrival distribution: We used the distribution of the number of flow arrivals per second given in [29] for our simulation. The paper fits a Poisson distributed curve to the trace and hence we used such a distribution for our flow arrivals. The number of YouTube servers (servers with unique IP addresses) used in the experiment was 2138. To scale our simulation we considered arrival rates to 1 and 10 servers. The experiment can simply be run for all servers with more powerful machines.

4) More Trace Experimental Results: To compare the performance of pure *Hincent* based approach against other TCP based approaches (PACE, Dandelion), we considered the best case scenario for the TCP based approaches. This scenario uses the *Hincent* content selection mechanism (see section XIII). So using this same server selection mechanism we compared the performance of the TCP-based approaches with our pure *Hincent* based approach. As can be seen from figures 12, 13 and 14, the pure *Hincent* approach gives lower file completion time when compared with TCP-based *Hincent* approach. For all experiments in this section, each YouTube file is divided into 50 chunks. So bigger file sizes have bigger chunk sizes. The YouTube video files we consider in this analysis are not live videos. Hence we use a content age of 15.5 seconds. This implies that videos which were first requested less than 15.5 seconds ago can still be requested. For all experiments of one YouTube server, the Intel i5 Core machine we used allowed us to run the simulation for 120 seconds. For the 10 YouTube servers experiments, we used a simulation time of 30 seconds.

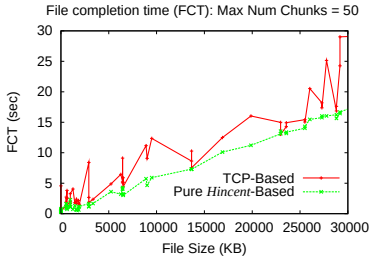


Fig. 12. File completion time with 1 YouTube server

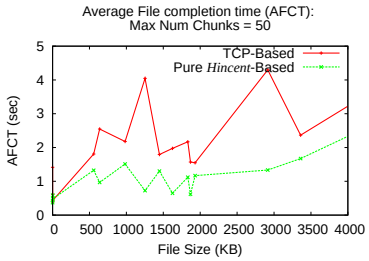


Fig. 13. Average file completion time (AFCT) with 1 YouTube server (small files)

Figure 13 shows the average file completion time (AFCT) of files less than 4000KB in size while figure 12 shows FCT of all files. As can be seen from figure 14, with more YouTube servers, the number of simulated peers requesting for content increases. This in turn increases the number of peers with a content and hence decreasing the file download time (AFCT). This is one of the noble gains of peer to peer systems as more peers means more bandwidth.

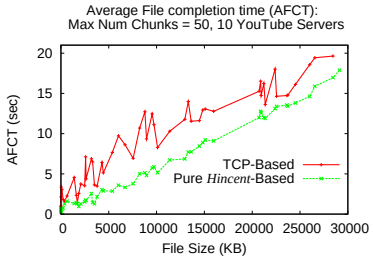


Fig. 14. Average file completion time (AFCT) with 10 YouTube server

Figures 15, 16 and 17 show that overwhelming majority of the peers do not have to spend money to download GB of data as the credit amount they earn balances out with the amount they pay. For each peer, the amount to spend in these plots is calculated as the total amount of money a peer earns minus the total amount a peer has to pay per GB of content.

Comparing figures 15 and 17, it can be seen that more YouTube servers in the experiment means more participating peers. The more peers have the contents the less other peers

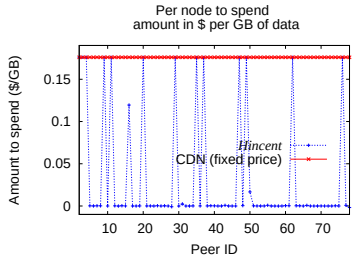


Fig. 15. Net amount to pay in dollars per GB of downloaded content with 1 YouTube server

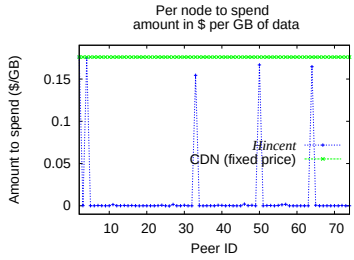


Fig. 16. Net amount to pay in dollars per GB of downloaded content with 10 YouTube servers (First few peers)

have to download the content from the CDN servers. This saves peers more money as can be seen from the plots. In all cases, the amount peers pay for bandwidth to download a content is less than the fixed CDN bandwidth amount charged by AmazonCLoudFront. For the experiments with only one YouTube server, the simulation generates fewer peers to download the content. As the number of peers which have the content is smaller, more peers download contents from the CDN servers paying more money as can be seen in figure 15. The amount which peers pay to directly download a content from the CDN servers can be subsidized (paid for) by the content providers as such peers are serving as seeders for the content provider.

F. CID Implementation Experiments

We have also implemented the basic features of *Hincent* in an Apache SQL server using PHP script. We implemented all the tables of the CID in an Ubuntu virtual machine using a quad four processor and a 1GB RAM. We generated *tblSelectedSource* table using a SELECT query from the tables *tblPeer*, *tblContent*, *tblPeerContent* as discussed in section VIII-A4 above. The tables are linked in a many-to-many relationship.

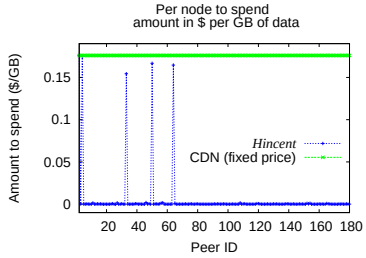


Fig. 17. Net amount to pay in dollars per GB of downloaded content with 10 YouTube servers (All peers)

To see the performance gain of using the *tblSelectedSource* table over generating the contents requested by peers on the fly from the three tables, we have conducted experiments using and not using the *tblSelectedSource* table. We used one million records in each table for this experiment. As can be seen from figures 18 and 19 preparing the *tblSelectedSource* table as its source tables are updated results in significant gain in query time. Here, query time is the time from when a query for a specific record is made to when the reply is displayed from the SQL server. In these experiments we first generated uniform random content index records with the given contentIDs to request from the SQL server. The content with the ID of *cont396224* was the first content requested. Such initial request of a record resulted in a higher query time perhaps because the SQL server took time to upload parts of the table into memory. Figure 19 also shows that the query time increases with the increase in the record ID. This is because the tables are roughly sorted by requestIDs as the none-numeric parts of the contentID and peerID values are the same while both fields have text data types.

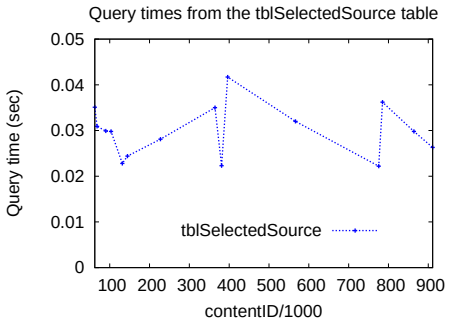


Fig. 18. Query time using the tblSelectedSource table

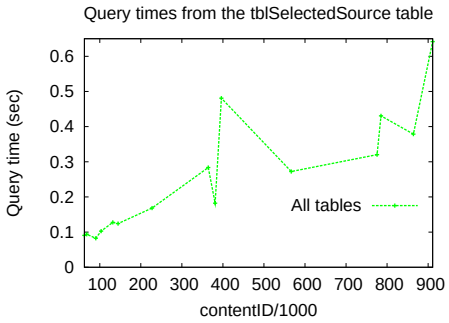


Fig. 19. Query time using SQL JOIN from all tables

The SQL query we made from the *tblSelectedSource* for the content with contentID of *cont396224* is as follows.

```
SELECT *
FROM ‘tblSelectedSource‘
WHERE contentID = ’cont396224’
```

```
LIMIT 0 , 1
```

And the following is the query we made from the three tables.

```
SELECT tblPeerContent.contentID, tblPeerContent.peerID,
  tblPeerContent.contentUrl, tblPeerContent.contentKey,
  tblContent.contentDesc, tblContent.contentPopularity,
  tblPeer.peerURate, tblPeer.peerUPrice, tblPeer.ratePerPrice
FROM tblPeerContent
INNER JOIN tblPeer ON tblPeerContent.peerID = tblPeer.peerID
INNER JOIN tblContent
  ON tblPeerContent.contentID = tblContent.contentID
WHERE tblPeerContent.contentID = 'cont396224'
LIMIT 0 , 1
```

We next conducted an experiment to know how long it takes for a query such as requesting the *contentKey* by a peer from the CID of the CIM. The propagation delay from the requesting peer virtual machine to the virtual machine with the SQL server is about 1ms. The times it takes for such query is shown in figure 20. There is a spike on the record of *cont132913* which is the first record requested by the peer in the experiment. Such a spike disappears with the other requested records as perhaps the SQL server caches the session and keeps the *tblRequestedContent* table loaded in memory.

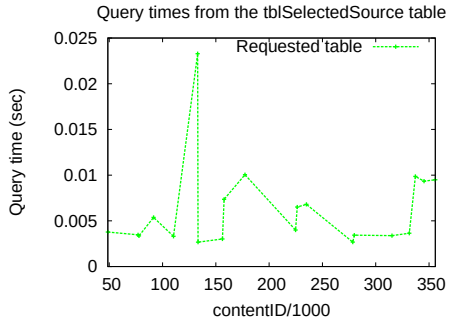


Fig. 20. Query time from peers to the CIM

The query we used for the experiments in figure 20 is as follows.

```
SELECT * FROM tblRequestedContent
WHERE contentID = 'cont$value'
LIMIT 0,1
```

These above query time figures are intended to show that the time it takes to resolve a certain query is not high even using a computer (server) with very limited hardward such as an Ubuntu virtual machine.

XIII. Related Work

Over time, Peer-to-peer (P2P) content distribution has evolved to incorporate incentives in order to prevent freeloading. The BitTorrent [30], [31] uses a rate based tit-for-tat mechanism where users can achieve higher download rate from peers to which they are uploading. In this case a peer which is not downloading a content is not incentivized to upload a content. In *Hincent* all peers are incentivized to continue uploading as every upload increases their credit maintained by the *Hincent* CIM. Reputation based schemes such as [32] help peers find another peer with the highest reputation score to download content from. Such a reputation scheme does not provide an accurate evaluation mechanism to choose a peer to serve a content. For instance a peer which is uploading many files without downloading a file can have a high reputation score. If such a peer does not have as much available upload capacity as another peer which is downloading files, peers will select it anyways because it has a high reputation score.

In the KARMA [33] scheme, every peer has a set of managers which form banks which coordinate credit transfer with other peers. In this scheme there is no guarantee of integrity of the global currency when the majority of the managers are malicious. In *Hincent* a central CIM which cannot be manipulated by peers offers real monetary rewards to all peers which upload contents. PACE [8] uses bandwidth pricing to help uploading peers earn credit. However PACE does not give a fair-exchange of content for payment as the content demand at a peer is estimated as a total requested download rate at remote buy clients. Such demand used to obtain a bandwidth price is not peer specific. Dandelion [3] is based on a centralized online currency bank mechanism to incentivize peers. However Dandelion uses a fixed pricing mechanism that peers are not awarded according to the upload bandwidth they offer to upload contents. Peers do not decrease their price to attract more customers when they have high upload rate and vice versa. PRIME [34] is a mesh-based P2P streaming. Even though it tries to balance the average outgoing rate of a source peer with the average incoming rate of a content receiving peer, it does not use an efficient rate allocation and enforcement mechanism like *Hincent* to achieve a max/min allocation. It uses a TCP friendly rate control protocol (TFRC) [35] which inherits the TCP problems of not quickly utilizing available link capacities. In PRIME each peer tries to maintain many parents that can collectively serve as content providers using a mesh-based overlay construction which can potentially incur significant overhead. Unlike *Hincent*, PRIME does not give an efficient mechanism to help peers select a content source with high throughput and minimum bandwidth cost. This is because a new peer selects a random subset of peers to be its content parents. A reliable client accounting system of a commercial hybrid content-distribution network (Akamai) is also presented in [36] to detect and mitigate a variety of attacks by malicious peers. This mechanism improves the NetSession which is a peer-assisted content delivery network (CDN) operated by Akamai. In *Hincent* peers do have any incentive to act maliciously. This is because peers get monetary incentives (credit) for uploading content and all transactions are co-ordinated by a scalable centralized *Hincent* CIM. If an *Hincent* peer acts maliciously, it only wastes its bandwidth and suffers monetary losses.

A hybrid CDN-P2P system for live video streaming called LiveSky is presented in [37]. The paper gives a trace based study of extensive LiveSky deployment in China. However the work only gives approximate guideline for peer selection. For instance the paper assume that the total upload bandwidth of clients in level k of the P2P tree is always larger than

the download bandwidth requirement of clients in level k+1. It also only considers aggregate measures (i.e., population and time averages) to model the end-user properties. On the other hand *Hincent* does not make such assumptions and uses accurate rate and price based incentives to select content sources to serve a content. This gives peers a reliable incentive to cooperate without a malice. LiveSky also limits peer selection to a local network while *Hincent* does not make that ristriction unless local content source selection stragy is used or the local peers have the best upload rate and lowest prices. A study in [38] shows that redirecting every client to the CDN server with least latency does not suffice to optimize client latencies. The authors of this paper proposed a system called *WhyHigh* to optimize Google CDN perfomance. WhyHigh measures client latencies across all nodes in the CDN and correlates measurements to identify the prefixes affected by inflated latencies. *Hincent* by design chooses peers or CDN servers which offer the highest throughput and lowest price and does not require complex inefficient systems such as WhyHigh to select content sources.

NetTube, a P2P assisted content delivering framework that explores the clustering in social networks for short video sharing is proposed in [39]. Like NetTube, *Hincent* allows users to share their contents while keeping it in their own servers. Unlike *Hincent*, NetTube selects a content source based on social groups and not based on throughput and bandwidth price. SocialTube, which is peer-assisted video sharing system that explores social relationship, interest similarity, and physical location between peers in online social networks (OSNs) is proposed in [40]. SocialTube uses a social network (SN)-based P2P overlay construction algorithm. Unlike *Hincent*, SocialTube does not select content sources based on a high upload bandwidth and low cost. This can result in SocialTube unnecessarily delaying streaming and other content transfer when other peers not in the same social group with high upload capacity exist.

Besides, unlike *Hincent*, all the above schemes do not help peers determine an accurate rate at which they can download content from other peers. They do not give a mechanism to prioritize content transfers which is an important component of 3D [10] and other streaming applications. Unlike *Hincent* they also do not provide an efficient max/min rate allocation mechanism.

XIV. Conclusion

In this paper we proposed the design of *Hincent*, an efficient cross-layer content routing and congestion control framework. Unlike existing content distribution approaches, *Hincent* relies on an accurate and fair incentive mechanism which allows prioritized max/min rate allocations and enforcements. *Hincent* is a flexible scheme which allows multiple server selection strategies. Unlike previous work we have presented a noble content index management scheme for *Hincent*. It allows distributed peers to have full control of their contents and to securely share them with others. We have also presented an extension of *Hincent* using surrogate servers with OpenFlow vSwitches to help peers exchange contents faster than using existing schemes.

We have implemented *Hincent* in the NS2 simulation package. We evaluated the performance of *Hincent* using rigorous trace based flow and packet level simulation experiments. The experiments demonstrate the *Hincent* design goals which result in lower content transfer time than existing schemes. We have also implemented *Hincent* content index management with Apache SQL server using PHP in Ubuntu virtual machines. The implementation experiments show that *Hincent* can easily scale to millions of content index records.

References

[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, Aug. 2010.

[2] C. Labovitz, "Internet traffic evolution 2007-2011," http://www.monkey.org/ labovit/papers/gpf_2011.pdf.

[3] M. Sirivianos, X. Yang, and S. Jarecki, "Robust and efficient incentives for cooperative content distribution," *IEEE/ACM Trans. Netw.*, vol. 17, pp. 1766–1779, Dec. 2009.

[4] P. Wendell and M. J. Freedman, "Going viral: flash crowds in an open cdn," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 549–558.

[5] C. Huang, A. Wang, J. Li, and K. W. Ross, "Understanding hybrid cdn-p2p: why limelight needs its own red swoosh," ser. NOSSDAV '08. New York, NY, USA: ACM, 2008, pp. 75–80.

[6] M. Marcon, B. Viswanath, M. Cha, and K. P. Gummadi, "Sharing social content from home: a measurement-driven feasibility study," ser. NOSSDAV '11. ACM, 2011, pp. 45–50.

[7] C. Aperjis, M. J. Freedman, and R. Johari, "Peer-assisted content distribution with prices," ser. ACM CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 17:1–17:12.

[8] C. Aperjis, R. Johari, and M. J. Freedman, "Bilateral and multilateral exchanges for peer-assisted content distribution," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1290–1303, Oct. 2011.

[9] M. Sirivianos, J. H. Park, X. Yang, and S. Jarecki, "Dandelion: cooperative content distribution with robust incentives," in *2007 USENIX Annual Technical Conference*, ser. ATC'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:14.

[10] Z. Yang, W. Wu, K. Nahrstedt, G. Kurillo, and R. Bajcsy, "Enabling multi-party 3d tele-immersive environments with viewcast," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, no. 2, pp. 7:1–7:30, Mar. 2010.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14 –23, oct.-dec. 2009.

[12] S. McCanne and S. Floyd, "NS-2," http://www.isi.edu/nsnam/ns/.

[13] D. Fesehaye and K. Nahrstedt, "Hincent: Quick Content Distribution With Priorities and High Incentives," in *IEEE Consumer Communications and Networking Conference (CCNC 2013)*, Las Vegas, USA, jan 2013.

[14] P. Pillay-Esnault., P. Moyer, J. Doyle, E. Ertekin, and M. Lundberg, "Ospfv3 as a provider edge to customer edge (pe-ce) routing protocol," United States, 2012.

[15] W. Wu, A. Arefin, Z. Huang, P. Agarwal, S. Shi, R. Rivas, and K. Nahrstedt, ""i'm the jedi!" - a case study of user experience in 3d tele-immersive gaming," ser. ISM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 220–227.

[16] Amazon, "Amazon cloudfront," http://aws.amazon.com/cloudfront, 2012.

[17] "The diaspora* project," http://diasporaproject.org/.

[18] K. Weise, "On diaspora's social network, you own your data," http://www.businessweek.com/articles/2012-05-10/on-diasporas-social-network-you-own-your-data, 2012.

[19] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[20] Internet2, "Internet2 Network: GlobalNOC RealTimeAtlas," http://atlas.grnoc.iu.edu/I2.html, 2012.

[21] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual switching in an era of advanced edges," Sep. 2010. [Online]. Available: http://openvswitch.org/

[22] B. Pfaff and et. al., "Openflow switch specification v1.3.0," https://www.opennetworking.org/, April 2012.

[23] D. Fesehaye, I. Gupta, and K. Nahrstedt, "A Cross-layer Routing and Congestion Control for Distributed Systems," University of Illinois at Urbana-Champaign (UIUC), TECHNICAL REPORT, Nov. 2008. [Online]. Available: https://www.ideals.illinois.edu/handle/2142/11503

[24] D. Fesehaye and K. Nahrstedt, "Finishing Flows Faster with A Quick congestion Control Protocol (QCP)," University of Illinois at Urbana-Champaign (UIUC), TECHNICAL REPORT, 01 2013. [Online]. Available: https://www.ideals.illinois.edu/handle/2142/35905

[25] P.-A. Larson, C. Clinciu, E. N. Hanson, A. Oks, S. L. Price, S. Rangarajan, A. Surna, and Q. Zhou, "Sql server column store indexes," ser. ACM SIGMOD '11. New York, NY, USA: ACM, 2011, pp. 1177–1184.

[26] "Planetlab 4hr traces," www.eecs.harvard.edu/syrah/nc/sim/pings.4hr.stamp.gz.

[27] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting video server selection strategies in the youtube cdn," ser. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 248–257.

[28] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, june 2008, pp. 229 –238.

[29] T. Mori, R. Kawahara, H. Hasegawa, and S. Shimogawa, "Characterizing traffic flows originating from large-scale video sharing services," ser. TMA'10. Springer-Verlag, 2010, pp. 17–31.

[30] B. Cohen, "Incentives build robustness in bittorrent," Proc. Workshop Econ. Peer-to-Peer Syst, 2003.

[31] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bit torrent," ser. NSDI'07. Berkeley, CA, USA: USENIX Association, 2007.

[32] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," ser. NOSSDAV '03. New York, NY, USA: ACM, 2003, pp. 144–152.

[33] V. Vishnumurthy, S. Chandrakumar, , and E. G. Sirer., "Karma: A secure economic framework for peer-to-peer resource sharing," Proc. Workshop on Economics of Peer-to-Peer Systems, 2003.

[34] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," *Networking, IEEE/ACM Transactions on*, vol. 17, no. 4, pp. 1052 –1065, aug. 2009.

[35] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "Tcp friendly rate control (tfrc): Protocol specification," United States, 2003.

[36] P. Aditya, M. Zhao, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, and B. Wishon, "Reliable client accounting for p2p-infrastructure hybrids," ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012.

[37] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *Proceedings of the 17th ACM international conference on Multimedia*, ser. MM '09. New York, NY, USA: ACM, 2009, pp. 25–34. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631279

[38] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving beyond end-to-end path information to optimize cdn performance," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 190–201. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644917

[39] X. Cheng and J. Liu, "Nettube: Exploring social networks for peer-to-peer short video sharing," in *INFOCOM*, 2009, pp. 1152–1160.

[40] Z. Li, H. Shen, H. Wang, G. Liu, and J. Li, "Socialtube: P2p-assisted video sharing in online social networks," in *INFOCOM*, 2012, pp. 2886–2890.